Department of Defense

High Level Architecture

Object Model Template

Version 1.1

12 February 1997

Table of Contents

List of Tables

FOREWORD

The formal definition of the Department of Defense High Level Architecture (HLA) comprises three main components: the HLA Rules, the HLA Interface Specification, and the HLA Object Model Template (OMT). This document is intended to provide a complete description of the essential elements of the third component of the HLA, the OMT. The other two components of the HLA formal definition are described in the following documents:

- **HLA Rules v1.0**
- **HLA Interface Specification v1.0**

In addition to these reference documents, the HLA Technical Library contains other information sources relevant to developing and executing HLA federations. The elements of the HLA Technical Library that are particularly relevant to HLA object model development include the following:

- **HLA OMT Extensions**: A description of the format and content of optional tables. These additional tables are intended to document classes of information which are not required for all HLA federations, but which may be useful for certain types of applications.
- **HLA Glossary**: A common set of semantics for terms used in the documents of the HLA formal definition or the HLA Technical Library.
- **HLA Federation Development and Execution Process Model**: A description of the process used to build and execute HLA federations.
- **HLA OMT Use Cases**: A set of case studies describing the process of developing HLA object models in different communities of interest. Each case study is based on the experiences of one of the HLA prototype federations (protofederations).
- **HLA OMT Test Procedures:** A set of procedures for testing compliance of an object model with the HLA OMT.

Other elements of the HLA Technical Library may also have some relevance to HLA object model construction. Users of this document are encouraged to browse the contents of the HLA Technical Library to discover sources of potentially relevant information, and to gain a broader understanding of other general HLA resources.

1. PURPOSE

The Department of Defense (DOD) Modeling and Simulation Master Plan [DOD95] calls for the establishment of a DOD-wide High Level Architecture (HLA) for modeling and simulation, applicable to a wide range of functional applications. The purpose of this architecture is to facilitate interoperability among simulations and promote reuse of simulations and their components.

To support the general goals of the HLA, this document provides a specification of the DOD HLA Object Model Template (OMT) for documenting key information about simulations and federations. More specifically, the HLA OMT provides a template for documenting HLA-relevant information about classes of simulation or federation objects and their attributes and interactions. This common template facilitates understanding and comparisons of different simulations and federations, and provides the format for a contract between members of a federation on the types of objects and interactions that will be supported across its multiple interoperating simulations. This document specifies both the type of information content required and a format for representing that content for HLA object models.

2. BACKGROUND

2.1 Object Model Template Rationale

A standardized structural framework, or template, for specifying HLA object models is an essential component of the HLA for the following reasons:

- Provides a commonly understood mechanism for specifying the exchange of public data and general coordination among members of a federation.
- Provides a common, standardized mechanism for describing the capabilities of potential federation members.
- Facilitates the design and application of common tool sets for development of HLA object models.

HLA object models may be used to describe an individual federation member (federate), creating an HLA Simulation Object Model (SOM), or to describe a named set of multiple interacting federates (federation), creating a Federation Object Model (FOM). In either case, the primary objective of the HLA Object Model Template (OMT) is to facilitate interoperability between simulations and reuse of simulation components. All discussion of HLA object models in this document applies to both SOMs and FOMs unless explicitly stated otherwise.

2.2 Federation Object Models

During development of an HLA federation, it is critical that all federation members achieve a common understanding as to the nature or character of all required interactions between participating federates. The primary purpose of an HLA FOM is to provide a specification of the exchange of all public data among federates in a common, standardized format. The content of this public data includes 1) an enumeration of all public object classes, 2) a description of all interaction types and associated parameters, and 3) a specification of the attributes that characterize the public objects. In addition, an HLA FOM may include supplemental information as described in the HLA OMT Extensions document. Taken together, the components of an HLA FOM establish the "information model contract" that is necessary (but not sufficient) to ensure interoperability among the federates.

2.3 Simulation Object Models

A critical step in the formation of a federation is the process of determining the composition of individual simulation systems to best meet the sponsor's overall objectives. An HLA SOM is a specification of the intrinsic capabilities that an individual simulation could offer to potential HLA federations. The standard format in which SOMs are expressed facilitates determination of the suitability of simulation systems for participation in a federation.

The HLA OMT formats described in this document are generally applicable to either FOMs or SOMs. Thus, SOMs are also characterized in terms of their objects, attributes, and interactions. The primary benefit from the common utilization of the OMT formats for FOMs and SOMs is that it provides a common frame of reference for describing object models in the HLA community. In some cases, this commonality may even allow SOM components to be integrated as “piece parts” in a FOM, facilitating rapid FOM construction.

2.4 Relation to Object-Oriented Object Models

While the HLA OMT is the standardized documentation structure for HLA object models, FOMs and SOMs do not correspond entirely to common definitions of object models in object-oriented (OO) development methodologies. The HLA object model combines elements of both the static and dynamic views of traditional OO object models. The static elements of an HLA object model include object classes, their attributes, and (optionally) associations, but do not currently include the object operations (or methods) of OO static models. The dynamic component of an HLA object model currently focuses on pairwise interactions between classes of objects, while OO dynamic models typically include additional information about sequences of events and state transition models of objects. Specification of HLA object class hierarchies tends to be driven by the interests of subscribing simulation systems, rather than the inheritance considerations that tend to dominate OO development models. HLA object models also differ in that they are ordinarily expected to contain less detail than an OO development object model since they are not designed for software development but for federation development.

Not only does the HLA conception of an object model differ from that of traditional OO object models, but HLA objects themselves also differ from the common OO conception of objects. Responsibility for updating HLA object attributes may be distributed among different federates in a federation, whereas OO objects characteristically associate update responsibilities with operations that are closely tied to the object’s class definition. This difference does not preclude OO implementations of objects within individual HLA federates; however, federation objects may transcend their individual representations within specific federates, being defined by the composition of all the attribute values published for them by any federate. When a federate instantiates an object, it initially owns those attributes of the object which it declared it would publish. However, ownership of some or all of these attributes may be transferred to other federates during the federation execution. When multiple federates own different attributes of the same object, responsibility for maintaining the object's state is effectively distributed across the federation, unlike a traditional OO object whose state is locally encapsulated.

In addition to the stated differences between HLA object models and traditional OO object models, there are also some differences in the semantics of the terminology used to describe similar concepts (e.g., class, object, interaction). Although descriptions of these concepts are provided later in this document, precise definitions of these terms can also be found in the separate HLA Glossary document.

3. HLA OMT COMPONENTS

HLA object models are composed of a group of interrelated components specifying information about classes of objects, their attributes, and their interactions. While it is possible to represent the information content of these components in many different ways, the HLA requires documentation of these components in the form of tables. The template for the core of an HLA object model uses a tabular format and consists of the following components:

- **Object Class Structure Table:** To record the subclass-superclass relations between different types of simulation/federation objects.
- **Object Interaction Table:** To record the types of interactions possible between different classes of objects, their affected attributes, and the interaction parameters.
- **Attribute/Parameter Table:** To specify features of the public attributes of objects and the parameters of interactions in a simulation/federation.
- **FOM/SOM Lexicon:** To define all of the terms used in the tables.

Both federations and individual simulations (federates) are required to use all four of the core OMT components when providing an HLA object model, although, in some cases, certain tables may be empty. Since all object information is classified by object classes, there must be at least one object class for any meaningful HLA object model. Thus, every HLA object model must have a Object Class Structure Table containing at least one object class.

While federations typically will support interactions among some of the objects of its federates, some federates (such as a stealth viewer) might not be involved in interactions, so the Object Interaction Table may be empty for some HLA object models. It is expected that federates will commonly have objects with attributes of interest across the federation, in which cases, their documentation in the Attribute/Parameter Table is required. However, a federate or an entire federation may exchange information solely via interactions, in which case its Attribute/Parameter Table may be empty. While either the Object Interaction Table or the Attribute/Parameter Table may, thus, be empty, an HLA object model would not be of much use if both of these tables were empty since such a model would not support any exchange of information between federates except for notifications of the existence of objects.

The final HLA OMT component, the FOM/SOM Lexicon, is essential to ensure that the semantics of the terms used in an HLA object model are understood and documented. Since there will always be at least one term in an HLA object model, there will always be at least one term defined in the Lexicon, and ordinarily many more.

Any entry within any of the OMT tables may be annotated with additional descriptive information outside of the immediate table structure. This "notes" feature permits users to associate explanatory information with individual table entries as required to facilitate effective use of the data. The format for attaching a note to a particular table entry is a numerical superscript enclosed by brackets. The note itself is identified by the corresponding superscript, and is unconstrained in terms of format. If a set of notes is defined for a given FOM or SOM, the notes must be included as part of the object model description.

In addition to the four OMT components identified above, federates and federations may also include supplemental categories of descriptive information in order to facilitate a more complete understanding of the object model. The format and content of this optional information is provided in the OMT Extensions Document.

The basics of each OMT component are presented in the following separate sections along with a brief review of the rationale for including them in the OMT. The template format for each component is provided and described. In addition, some criteria are suggested to help guide decisions on when to include specific simulation or federation features within each of these components for a specific HLA object model.

3.1 Object Class Structure Table

3.1.1 Purpose/Rationale

The object class structure of an HLA object model is defined by a set of relations between classes of objects from the simulation or federation domain. An HLA object model class is a collection of objects with some properties, behavior, relationships, and semantics in common. Each of the individual objects in a class is said to be a member (or instance) of that class. Class names in an HLA object model must be defined via the ASCII character set, and must be globally unique: no class name in a Class Structure Table may be identical to any other class name elsewhere in this table. However, class names may include other class names as parts (textual substrings) to indicate relations between classes.

An HLA class structure is defined in terms of hierarchical relationships between classes of objects. Immediate superclass-to-subclass relationships are represented via the inclusion of the associated class names in adjacent columns of the Object Class Structure Table. Non-immediate superclass-to-subclass relationships are derived via transitivity from the immediate relations: if A is a superclass of B, and B is a superclass of C, then A is a (derived) superclass of C. Superclass and subclass play inverse roles in these relations: if A is a superclass of B, then B is a subclass of A.

Subclasses can be considered to be specializations, or refinements, of their immediate superclasses. Subclasses always inherit the characteristics (attributes and interactions) of their immediate superclass, and may possess additional characteristics to provide the desired specialization. These types of object class relationships (referred to as “is-a” relationships in the OO literature) may also be defined in terms of their instances: a class A is a superclass of class B only if each of the instances of class B are also instances of class A. Under this conception, it is useful to distinguish derived instances of a class from explicitly declared instances. Once an object is explicitly declared to be an instance of some object class, it becomes an implicit (or derived) instance of all the superclasses of that class. For example, if the class M1_Tank is a subclass of Tank, then an object declared to be an M1_Tank, will be a derived instance of Tank. While some classes (such as Tank) might be designed for organizational purposes and not intended to have any explicitly declared instances, such "abstract" classes may still have derived instances.

A class is a root in a class structure if it has no superclasses in that structure. A class is a leaf of a class structure if it has no subclasses. If each class has at most one immediate superclass, then the class structure is said to have single inheritance and will form either a tree structure or a forest of trees, depending upon whether there are one or more roots. If some classes have more than one immediate superclass, then the class hierarchy is said to have multiple inheritance. HLA object model class hierarchies must be represented via single inheritance (no multiple inheritance), although flat structures (with no subclasses) are also permissable.

In general, simulations and other federates participating in a federation execution may subscribe to object classes at any level of the class hierarchy. By subscribing to all attributes of a specified object class, a federate is ensured of receiving all attribute value updates of attributes defined for that class and all of its superclasses for all instances of that class and all instances of its subclasses. After subscribing to an object class, a federate is notified by the *Discover Object* service of the Run-Time Infrastructure (RTI) of the existence of any instances of that class (or its subclasses) which meet their discovery criteria. This service provides a class name and object ID (plus any available attribute values) for every such discovered object. The RTI will report objects as belonging to the most specific object class or classes to which the federate is directly subscribed and which meets the federate’s discovery criteria. If the federate subscribes to multiple levels, the RTI’s discovery notification will identify an object as an instance of the lowest-level class (or classes) to which the object belongs among those subscribed by the federate.

Object classes provide the means for federation participants to subscribe to information about all individual instances of objects with common characteristics, such as all M1A1 tanks or

F117A fighters. Classes are also essential to specifying the types of attributes and interactions characteristic of simulation objects since these are defined relative to classes of objects, not unique to individual instances. Basic services of the HLA RTI support subscriptions to object classes and their attributes by federates participating in a federation execution. So the RTI needs to know the object classes, attributes, and interactions if it is to perform consistency checks and to support distribution of object information by class to the federates of a federation execution.

A class hierarchy expands the capabilities of a flat classification scheme by enabling federates to subscribe to information about broad superclasses of objects, such as all tanks, all attack fighters, or even all ground vehicles, air vehicles, or sea vehicles. The existence of a class hierarchy can simplify the subscription to class information when federates are interested in broad classes of objects. The HLA interface supports subscription to all attributes of any class in an object class hierarchy so that federates can easily subscribe to all and only those classes of interest. An object class hierarchy also supports simplification of the specification of attributes, by placing common attributes of multiple subclasses in a common superclass. Thus, class hierarchies enable simpler management of the interests of different federates in the objects and attributes involved in a federation execution.

The interest management simplification enabled by object class hierarchies also extends to interests in interactions. An object class hierarchy supports modeling of interactions at multiple levels of specificity with respect to the classes of interacting objects since the objects in a class inherit the interactions of their superclasses. For example, a weapon fire interaction might be specified as a single relation between any two objects in the platform class, rather than specifying a separate interaction type for every specific pair of platform subclasses. Thus, object class hierarchies enable specification of interaction hierarchies, which permit subscription to interactions at levels appropriate to a federate's interests.

3.1.2 Table Format

The object class structure template of Table 3-1 provides a format for representing the subclass-superclass hierarchy of object classes. It begins with the most general object classes in the left column, followed by all their subclasses in the next column, and then a further level of subclasses. The number of intermediate columns used here depends upon the needs of the federation. A federation that uses a deeper hierarchy than illustrated by the template of Table 3-1 may add columns as needed. Finally, the most specific object classes are specified by enumeration in the farthest right column. For cases in which the whole class hierarchy is too

		Structure Table	
		...	...
		...	...
<class> (<ps>)	...	...	...
	[<class> (<ps>)]	[<class> (<ps>)]	
		[<class> (<ps>)]	
		...	
...	...	...	... ...
	[<class> (<ps>)]	[<class> (<ps>)]	
		[<class> (<ps>)]	
		[<class> (<ps>)]	
	[<class> (<ps>)]	[<class> (<ps>)]	

Table 3-1. Object Class Structure Table

deep to fit across a single page, a reference (<ref>) to a continuation table may be provided in the last column. Each object class must have all of its subclasses specified in the next column to its right or in a continuation table referenced in that column. An example of how such a table may be filled in is provided in Section 3.1.4, as well as in the separate HLA OMT Use Cases document. See Appendix A for a brief description of the notation that is used for specifying entries for this table.Each object class in the Object Class Structure Table will be followed by information on publication and subscription capabilities enclosed in parentheses, as designated in the template using the abbreviated variable name <*ps*>. Three basic capability levels are distinguished with respect to a given object class:

publishable (P): The specified object class can be published by a federate using the *Publish Object Class* service of the RTI. This also requires that a federate is capable of meaningful invocations of the *Register Object* service of the RTI using this class's name.

subscribable (S): A federate is currently capable of utilizing and (potentially) reacting to information on objects in the specified class. Qualifying for this subscription category requires the minimal capability of being able to respond appropriately to the RTI message of *Discover Object* for objects of this class.

neither publishable or subscribable (N): The object class is neither publishable nor subscribable by a federate.

These definitions apply equally to FOMs and SOMs, although an object class only needs to be publishable or subscribable by a single federate in a federation for it to be classified as publishable or subscribable, respectively, by the federation as a whole.

The *publishable* and *subscribable* capabilities are intended to identify meaningful capabilities of a federation or federate with respect to the associated object classes. Although it is difficult to formulate precise criteria for distinguishing such capabilities for all possible cases, the general intended interpretation may be characterized. An object class is publishable by a federate in this sense only if the federate is capable of somehow modeling the existence of objects in this class when it instantiates them. It is not enough to be capable of issuing calls to the cited RTI services for publication or instantiation, which any simulation might easily accomplish for any arbitrary object class. The *publishable* designation is intended to allow federates to distinguish their internal capabilities for modeling objects of the associated classes as well as their ability to share information about such objects in an HLA federation. An object class is *subscribable* by a federate only if the federate can make substantive use of instances of the class when it is notified of them by the RTI. An object class is not *subscribable* by a federate if it always ignores instantiation notices and updates for object attributes in that class. While the HLA requires that substantive capabilities underlie designations of object classes as *publishable* or *subscribable*, the detailed determination of what is meant by "substantive" for a particular FOM or SOM must be left to the discretion of their developers.

The *publishable* and *subscribable* capabilities may both be present for an object class, or various other combinations, depending on the type of class. Classes that are not *publishable* may be "abstract". An abstract class has no explicitly declared instances since instantiations using its class name are not permitted. However, abstract classes ordinarily have "concrete" subclasses, i.e., subclasses which can be instantiated. Abstract classes can be useful for subscription purposes, simplifying some subscriptions to information about objects in their subclasses. Abstract classes can also simplify the specification of attributes by allowing common attributes of multiple object classes to be specified once in a common abstract superclass.

An individual federate must specify its publishing and subscription capabilities in its SOM Object Class Structure Table by any of the four different combinations of publishing and subscription capabilities from the set {P, S, PS, N}. An object class may be *publishable* without being *subscribable* (*P*), may be *subscribable* without being *publishable* (*S*), or may both *publishable* and *subscribable* (*PS*) for an individual federate. In some cases, a federate may even have an abstract object class in its SOM which is neither *publishable* nor *subscribable* (*N*). Such an object class might be included in a SOM to provide a convenient grouping of concrete subclasses for purposes of defining an interaction which could be initiated by an instance of any of these subclasses. To illustrate, an object class of *Ground_Vehicle* might be abstract, not published, and not *subscribable*, but could provide a convenient means of defining a *Ground_to_Air_Engagement* interaction (which is publishable and subscribable). Without such general classes, a *Ground_to_Air_Engagement* could not be so succinctly defined as an interaction between objects in the classes of *Ground_Vehicle* and *Air_Vehicle*.

Publication and subscription capabilities for a federation are somewhat different from those of a single federate. Whenever a federation supports publication of an object class, it must support subscription as well since it would be useless to publish an object class that could not be subscribed to within a federation. Thus, the *publishable/subscribable* capability designations for an object class in a FOM are taken from the more restricted set {*S*, *PS*, *N*}. This allows the publication and subscription capabilities recorded in a FOM to distinguish between abstract classes *(S) or (N),* and concrete, publishable and subscribable *(PS)* classes.

3.1.3 Inclusion Criteria

The criteria for designing a object class hierarchy for an HLA object model are fundamentally different for individual federates than for federations. The Object Class Structure Table of a FOM represents an agreement between the federates in a federation on how to classify public objects for the purposes of federation executions. The Object Class Structure Table of a SOM is a type of advertisement of the classes of objects which the federate can support (as publishable or subscribable) in potential federations. In neither case does the HLA require specific object classes or object class hierarchies to appear in the Object Class Structure Table. However, reference to an object class in another component (table) of a FOM or SOM always requires its inclusion in the Object Class Structure Table.

For federations, multiple criteria can influence the construction of a suitable object class hierarchy. Naturally, there must be classes for all of the types of objects that will participate publicly in federation executions. An object is understood as participating publicly in a federation execution whenever any of its attributes or interactions are being published during that execution. Thus, it is a federation's interests in information about simulation objects, their attributes, and interactions that drive which classes should be included in a federation's Object Class Structure Table.

Three general criteria have been identified that require a suitable class to be included within the object class hierarchy of an HLA FOM:

- Publication of object attributes.
- Publication of object interactions.
- Subscription to attributes or interactions at a higher level of object class abstraction.

Published attributes are those whose values a federate makes available to other federates during execution. An object class must be listed in the Object Class Structure Table for any object with public attributes since objects can be instantiated through the RTI during a federation execution only if they are associated with a class. The initiating object class(es) and receiving object class(es) (if any) of every HLA interaction must also appear in the Object Class Structure Table. In fact, any object class that is referenced elsewhere in an HLA object model (including any optional information as described in the HLA OMT Extensions document) must also be included in the object class hierarchy.

While a set of concrete object classes for the most specific types of entities involved in a federation (e.g., M1 tanks and Bradley fighting vehicles) may completely satisfy the subscription requirements of some types of HLA applications, additional higher-level object classes will be needed if federates wish to be able to subscribe to object information at higher levels of abstraction (e.g., tanks, armored vehicles, or ground vehicles). For a federate to be able to subscribe to object information at a desired level of abstraction, an object class at that level of abstraction must appear in the Object Class Structure Table. For example, suppose a federation involved both air, land, and sea forces of many specific types. If a particular federate did not require notification of the specific types of land vehicles, but did require notification of land vehicles in its area of interest, then a suitable abstract class (such as *Ground_Vehicle*) would be needed to make this possible.

While classes are clearly needed for all the public objects, many alternative class hierarchies can be devised to cover any given set of objects. The particular demarcations and levels of classes selected for an HLA FOM are the result of the federation development process. This selection is driven largely by the interests of the federates in subscribing to information about classes of objects. Object class hierarchies that may already exist for individual simulations may be incorporated into a FOM object class hierarchy if they meet the interests of the federation as a whole. However, since new classifications of objects may be warranted to meet federation needs which were not previously made explicit in any of their participating federates, FOM object classes and their subclass relations are not constrained to be a subset of those of the SOMs of the participating federates.

3.1.4 Example

Table 3-2 illustrates an example of how the Object Class Structure Table may be utilized to represent a simple system. In this case, the system being represented is a typical neighborhood restaurant. The simulation of this restaurant's operations can be considered to be a potential federate in a larger-scale federation, perhaps representing the combined, coordinated operation of a chain of restaurants. The intent of this example is not to specify a complete SOM for this system, but rather to provide partial illustrations as to how the OMT tables may be used to capture relevant information about the system.

In this example, a subset of a complete object class hierarchy is shown as consisting of five object classes at the uppermost level. For this particular simulation, no class decomposition was necessary for the first three classes. For the fourth class, a single level of decomposition is shown resulting in five leaf classes. For the fifth class, several levels of decomposition are shown to illustrate a partial representation of the restaurant's "menu". Some of the deeper levels in this hierarchy could have been modeled as attributes (e.g., *Clam_Chowder* could have been a leaf node, with an attribute of *Type* to represent the enumerated values of *Manhattan* or *New_England*). However, the modeler in this example opted to represent the most specific food types as individual classes. In all cases in this example, abstract classes are designated as "subscribable" only, while the leaf nodes (concrete classes) are designated as both "publishable" and "subscribable".

Object Class Structure Table				
Customer (PS)				
Bill (PS)				
Order (PS)				
Employee (S)	Greeter (PS)			
	Waiter (PS)			
	Cashier (PS)			
	Dishwasher (PS)			
	Cook (PS)			
Food (S)	Main_Course (PS)			
	Drink (S)	Water (PS)		
		Coffee (PS)		
		Soda (S)	Cola (PS)	
			Orange (PS)	
			Root_Beer (PS)	
	Appetizer (S)	Soup (S)	Clam_Chowder (S)	Manhattan (PS)
				New_England (PS)
			Beef_Barley (PS)	
		Nachos (PS)		
	Entree(S)	Beef (PS)		
		Chicken (PS)		
		Seafood (S)	Fish (PS)	
			Shrimp (PS)	
			Lobster (PS)	
		Pasta (PS)		
	Side_Dish(S)	Corn (PS)		
		Broccoli (PS)		
		Baked Potato (PS)		
	Dessert (S)	Cake (PS)		
		Ice_Cream (S)	Chocolate (PS)	
			Vanilla (PS)	

Table 3-2. Object Class Structure Table - SOM Example

3.2 Object Interaction Table

3.2.1 Purpose/Rationale

An interaction is an explicit action taken by an object that can optionally be directed toward another object, geographical area, etc. Interactions are specified in the Object Interaction Table of HLA object models in terms of the interaction structure, the classes of the initiating and receiving objects, their affected attributes, and the parameters of the interaction. In addition, the capabilities of an individual federate for initiating, sensing, and reacting to such interactions are recorded for SOMs.

The interaction structure of an HLA object model is a hierarchical structure composed of relations of generalization (or specialization) between different types of interactions. For example, an engagement interaction might be specialized by air-to-ground engagements, ship-to-air engagements, and others. This engagement interaction, then, would be said to generalize its more specific types. If there are no generalizations of interaction types for a federation or simulation, then the interaction structure will be flat, consisting of a set of unstructured interactions.

An interaction hierarchy in an HLA object model is designed to support inheritance in subscriptions. When a federate subscribes to an interaction class, using the *Subscribe Interaction Class* service of the RTI, it receives notification of all interactions that occur during a federation execution which are identified as instances of that class or as instances of any of its subclasses. Subscribing to an engagement interaction, for example, would result in notification of all air-to-ground engagements and ship-to-air engagements if they are subclasses of this interaction.

The classes of objects involved in interactions of a specific type are identified along with the type. Such classes may be designated for initiating objects as well as receiving objects. Initiating objects are those to which credit may be given for initiating an interaction, while receiving objects may be affected by the interaction but do not initiate it. A common example for initiating and receiving object classes comes from weapon engagement interactions, in which the source of the weapons fire is the initiating object and the recipient of the weapons fire is the receiving object. More specifically, in an air-to-ground engagement, the initiating object class might be that of *Air_Vehicle* while the receiving object classes might include *Ground_Vehicle* and *Cultural_Feature.*

While both initiating and receiving object classes are accommodated by the HLA OMT, there may be no specific receiving object class in some cases. For instance, if a platform is firing into an area instead of at a specific target, its responsible federate simply sends the firing event (interaction) to the current federation execution along with associated parameters. All federates with objects that may be affected by this event (e.g., objects in the area targeted) simply subscribe to that particular interaction class, and decide for themselves whether instances affect them or not. Thus, for some classes of interactions, the receiving object field of the HLA OMT interaction component will make sense (e.g., weapon detonates at platform) while for interaction classes in which there is no clear recipient, it may not. Initiating object classes are always required for interactions in the Object Interaction Table, although the initiating class may be notional (rather than explicitly known) in SOMs when the simulation can receive but not initiate the interaction.

Furthermore, the distinctions of initiating and receiving objects may not be relevant to some types of interactions. Some types of collisions, for example, in which two independently moving objects collide, may provide no basis for distinguishing an initiating and receiving object. For any such interactions, the HLA OMT is indifferent about which classes of involved objects are designated as initiating and which are designated as receiving in the HLA object model.

Included with the initiating and receiving object classes designated for each type of interaction are those attributes of these objects which may be affected by the interaction, along with optional comments on the nature of the effects. Not all interactions will affect attributes of both initiating and receiving object classes. The receiving object class most commonly has its attributes affected by an interaction, either directly by a change in attribute value, or indirectly by influencing future variations in that attribute's value. An air-to-ground weapons engagement, for example, might affect the location of a tank indirectly by immobilizing it, although it does not change the current value of its location. All such affected attributes, whether directly altered or indirectly affected, should be documented with the interaction in the HLA object model. Additionally, the object model may use square brackets to distinguish those attributes that are only possibly affected by an interaction from those that are always affected by a particular type of interaction. Comments may be included to clarify the types of effects possible (e.g., immobilization) but the detailed algorithms determining those effects are not currently included in HLA object models.

Interaction parameters in HLA object models record the parameters of an interaction. These parameters are precisely those that are sent along with the interaction class name and interacting object IDs in a call to the *Send Interaction* service of the RTI. Examples of interaction parameters include object class names, object attributes, constants, and other user-defined datatypes. Interaction parameters may be required to specify some features or properties of an interaction which are needed to calculate its effects by a receiving object. The HLA object model should only include those interaction parameters which are intended to be passed through the RTI *Send Interaction* service. The names of all such interaction parameters are documented in the Object Interaction Table. Details on these parameters, such as resolution and accuracy, may be found in the Attribute/Parameter Table of an HLA object model. Interaction parameters are specified separately for each interaction class in the hierarchy and are not inherited from interaction superclasses.

Interactions are one of the principal determinants of interoperability between simulations. Interoperability ordinarily requires some consistency in the treatment of interactions afforded by the different federates in which they appear. Consistency among the federates of a federation requires consistent responses to the same types of public (or cross-federate) interactions, regardless of who owns the initiating or affected objects. In distributed war fighting, for example, some uniformity in treatment of engagement interactions is commonly required to ensure a fair fight between objects owned by different federates. Thus, it is essential that all public interactions in a FOM be identified and that all federates in an HLA federation treat the specified interactions in a uniform fashion.

In addition, the RTI must know the types of interactions involved in a simulation execution in order to support publication and subscription to their occurrences. Thus, the HLA object model must document all of the interactions that may be sent during a federation execution so that the RTI can recognize them. Inclusion of the initiating and receiving object class types in the object model facilitates determination of all those federates in a federation that must directly accommodate a particular interaction, since different federates typically support ownership of different object classes. Inclusion of the parameters of interactions in the object model serves to identify the specific parameters that may be provided by any federate sending this interaction, and responded to by any federate whose objects are recipients of its effects.

3.2.2 Table Format

The template for recording object interactions for a federation or an individual federate is illustrated in Table 3-3. It contains five main sections: interaction structure; initiating object class information; receiving object class information; interaction parameters; and the capabilities of a federate to initiate, sense, and/or react to the interaction. For any interaction in the table, an interaction type (name) and initiating object class must always be specified. Interaction names in an HLA object model must be defined via the ASCII character set, and must be globally unique: no interaction name in an Object Interaction Table may be identical to any other interaction name elsewhere in this table. A receiving object class need not be specified in cases where no special recipients are identified. The interaction parameters may be empty if there is no need for the information they might supply. The initiates/senses/reacts capabilities of a federate for each interaction class should always be specified for SOMs. FOMs must also include this class of information for uniformity.

The interaction structure is shown in Table 3-3 with two columns that are intended to capture some of the structure of interaction classes. The first column lists the most general type of interaction, while the second column lists more specific interactions of the type of its corresponding first column. If there is no hierarchical structure to interactions, then this second column is unnecessary. If a simulation or federation has a deeper structure for interactions, dot notation should be used in the interaction name of the first column to capture all the structure

Interaction Structure			Initiating Object		Receiving Object/Area		Interaction Parameters	Init/ Sense/ React
			Class	Affected Attributes	Class	Affected Attributes		
<interaction>			<class> [,<class>]*	[<attribute>] [,<attribute>]* [(<comment>)]*	[<class>] [,<class>]*	[<attribute>] [,<attribute>]* [(<comment>)]*	[<parameter>] [,<parameter>]*	<isr>
...	...	...	...	...	...	...	...	...
	[<interaction>]		<class> [,<class>]*	[<attribute>] [,<attribute>]* [(<comment>)]*	[<class>] [,<class>]*	[<attribute>] [,<attribute>]* [(<comment>)]*	[<parameter>] [,<parameter>]*	<isr>
...	...		...	...	...	...	...	...
<interaction>	[<interaction>]		<class> [,<class>]*	[<attribute>]		[<attribute>]		<isr>

Table 3-3 Object Interaction Table

down to the finest level of interaction. For example, if there is a sequence of interaction subtypes, (interaction$_1$, interaction$_{1-1}$, interaction$_{1-1-3}$), where interaction$_{1-1-3}$ is a subtype of interaction$_{1-1}$ which is a subtype of interaction$_1$, then the compound entry "interaction$_1$. interaction$_{1-1}$" may appear in the first column of the interaction structure, while the second column has the simple entry "interaction$_{1-1-3}$". This convention helps conserve horizontal space in an already cramped table. See Appendix A for a brief description of the general format used for specifying entries in this table template.

The next group of columns lists the classes of objects involved in the specified interaction and those of their public attributes which may be affected by the interaction. Initiating and receiving objects are distinguished. If there is no basis for distinguishing an initiating from a receiving object, either of the main participating object classes may be placed in either the initiating or receiving columns. Either initiating or receiving objects may be specified by more than one class in cases where multiple classes participate in the same manner in an interaction but no common superclass shares that participation. The attributes of the interacting object classes that are potentially affected by the interaction should be listed following the object class. Comments may be included to clarify the nature of the effects on attributes, as indicated.

The parameter column lists the parameters of an interaction. These are the same parameters that appear in a call to the *Send Interaction* service of the RTI for the listed interaction. If no parameters are ever required for a particular type of interaction, then *N/A* should be entered in the parameter column to indicate this.

The primary intent of the Init/Sense/React column of the Object Interaction Table is to categorize the current capabilities of an individual federate with respect to object interactions. Three basic categories are used to indicate capabilities with respect to a given type of interaction:

initiates (I): indicates that a federate is currently capable of initiating and sending interactions of the type specified in that row of the Object Interaction Table.

senses (S): indicates that a federate is currently capable of subscribing to the interaction and utilizing the interaction information, without necessarily being able to effect the appropriate changes to affected objects.

reacts (R): indicates that a federate is currently capable of subscribing and properly reacting to interactions of the type specified by effecting the appropriate changes to any owned attributes of affected objects.

A capability of *initiates* for an interaction requires not just the ability to call the HLA *Publish Interaction Class* service for that interaction, but also the ability to model the initiation of the interaction and to invoke the HLA *Send Interaction* service for such interactions when initiated.

A federate *senses* a class of interactions if it is capable of utilizing information about such interactions via a *Receive Interaction* message after having invoked the *Subscribe Interaction Class* service of the RTI. It is not enough to simply be capable of receiving such interaction messages, which any HLA compliant federate may do, but the information received in such messages must be used somehow by the federate. For example, a stealth viewer that is incapable of determining the effects of interactions might subscribe to them in order to adjust its display accordingly (e.g., to show flashes during weapons fire). Such a viewer *senses* these types of interactions, even though it never *reacts* to them, as described next.

A federate *reacts* to a class of interactions only if it has the capability for owning the object ID of objects in the receiving class and/or has the capability for publishing affected attributes of receiving objects. In this latter case, the federate must also be capable of updating the values of those attributes to properly reflect the effects of the interaction. Naturally, not all interactions may require changes to attribute values, but instead may involve changes to internal states that affect attribute value updates. Minimally, a *reacts* capability for an interaction class requires a federate's ability to respond appropriately to the *Receive Interaction* calls from the RTI for such interactions. Appropriate response capabilities include the ability to alter future updates of some of the affected attributes, i.e., to affect the behavior of the affected objects.

Merely being able to reflect changes to the attribute values of objects affected by an interaction does not represent a *reacts* capability for the interaction. A simulation that simply reflects the consequences of some interaction in virtue of reflecting changes to the attribute values of its affected objects without being able to generate such changes itself is described as reflecting the attribute, not reacting to the interaction.

In a federation, at least one federate must have an *initiates* capability and at least one federate must have either a *senses* or a *reacts* capability in order for an interaction to be included in the FOM. Thus, a federation will always support one of the combinations *IS* or *IR* for each interaction. An individual federate may support several more combinations of initiating, sensing, and reacting to an interaction: {I, S, R, IS, or IR}. Any interaction in the SOM of a federate must have one of these combinations of Init/Sense/React capabilities. If a federate cannot either initiate, sense, or react to an interaction, then that type of interaction does not belong in its SOM.

3.2.3 Inclusion Criteria

A type of interaction should be included in a FOM whenever it can take place "across" a federation, i.e., when it is an "external" type of interaction. Common examples of such interactions in warfighting simulations include a variety of engagement interactions between platforms which may be owned by different federates. It is essential for a FOM to include all external interactions in order to document the types of interactions that federation members and the RTI may need to accommodate.

When interactions are not expected to occur across a federation, they need not appear in an HLA FOM. For example, the interactions involved in the internal dynamics of an engine in an engineering simulation of a vehicle might not be part of a FOM if no other federate in the federation will interact directly with the engine component.

Since HLA SOMs are intended to be developed independently of any particular federation application, the particular relevance of any currently supported interaction class to future federations will generally be unknown. Thus, a simulation which supports either initiating, sensing, or reacting for an interaction class should ordinarily document that support in its SOM if it is considered of possible interest to future federations.

3.2.4 Example

A representation of some illustrative interactions, based on the restaurant example introduced in Section 3.1.4, is given in Table 3-4. Here, two different abstract interaction classes are each decomposed into two lower-level classes. In the first case, the abstract *Food_Arrives* interaction class is decomposed into the *Food_Arrives_at_Waiter* and *Food_Arrives_at_Customer* interaction classes. The first of these is represented as an interaction between a *Cook* object class and a *Waiter* object class, and is meant to represent the event that the cook has finished preparing the order, and is now handing off the order to the waiter for delivery. For the cook, an attribute named *Orders_Pending* will be reduced by one due to this interaction, while the waiter may have his *State* attribute modified to reflect his next task of delivering the food. The two interaction parameters *Order_Number* and *Table_Number* are required by the *Waiter* object class to perform required operations in his new state. Finally, as in all of the interactions illustrated in this example, it is assumed that the restaurant federate can both initiate and react to interactions of this type.

The interaction *Food_Arrives_at_Customer* is meant to represent the arrival of the food at the customer’s table. This may trigger another change of *State* for the waiter, and will affect the degree of *Satisfaction* the customer has with the meal depending on the values of the three interaction parameters.

Interaction
Parameters
Init/Sense
/React
IR
IR
IR
IR
Class
Cook
State
Class
Waiter
State
Waiter
N/A
at_Waiter
Customer
Food_Arrives
Pay_Bill
Cashier
Satisfaction
Food_Arrives_
Pay_Bill_
Customer
Order_Number,
Table_Number
Cashier
at_Customer
Orders_Pending
(Reduce by 1)
Food_Arrives_
Pay_Bill_
By_Cash
Customer
Accuracy_OK,
Bill_Amount,
Timeliness_OK
Card_Validity
Bill_Amount
Temperature_OK,
Affected Attributes
Initiating Object
Affected Attributes
Daily_Receipts,
by_Credit_Card

Table 3-4. Object Interaction Table - SOM Example

The abstract *Pay_Bill* interaction class is decomposed according to whether the customer is paying by cash or by credit card. Although each involves an interaction between the customer and the cashier, different attributes may be affected (as shown). In addition to specifying the bill amount as an interaction parameter, the validity of the credit card (e.g., expiration date, credit limit) may be needed for credit card interactions to determine if the transaction can be successfully executed.

3.3 Attribute/Parameter Table

3.3.1 Purpose/Rationale

Each class of simulation domain objects is characterized by a fixed set of attribute types. These attributes are named portions of their object’s state whose values can change over time (such as location or velocity of a platform). Public attributes are those domain object attributes whose values may be published through the RTI and provided to other federates in a federation. An HLA FOM documents all such public attributes in the Attribute/Parameter Table. Because of the similarity between information about attributes and interaction parameters, all interaction parameters are documented along with attributes in the same table.

An HLA object model supports representation of the following characteristics for attributes in the basic Attribute/Parameter Table:

- Object class
- Attribute name
- Datatype
- Cardinality
- Units
- Resolution
- Accuracy
- Accuracy condition
- Update type
- Update rate/Condition
- Transferable/Acceptable
- Updateable/Reflectable

The object class specifies the class of objects to which the attribute applies. The attribute name identifies the attribute. The datatype column specifies the datatype of each attribute. The units entries identify the units (such as m, km, kg) used for attribute values. A resolution characteristic is intended to record how finely the published values of an attribute may differ from each other. When attribute values take numeric values, a minimum possible quantitative variation in attribute value may be recorded here. When attribute values are discrete, then this fact may be recorded.

The accuracy of an attribute captures the maximum deviation of the attribute value from its intended value in the simulation or federation. This is often expressed as a numeric value, but may also be *perfect* for attributes which have no deviation from intended values. The accuracy condition of an attribute specifies any conditions required for the given accuracy to hold at any given time during simulation/federation execution. It may consist of a reference to a particular type of update algorithm that determines the accuracy, or may be an unconditional *always.*

The update type and update condition characteristics specify the update policies for the attribute. The transferable/acceptable characteristic provides an indication of whether ownership of the attribute can be transferred to or accepted from different federates. Finally, the updateable/reflectable characteristic is used to indicate capabilities for updating and reflecting the attribute.

Interaction parameters are characterized in much the same way as attributes. Minor differences include the fact that the interaction class is listed instead of the object class, and the parameter name is listed instead of the attribute name. A more significant difference is that parameters only utilize the first eight characteristics (object class, attribute name, datatype, cardinality, units, resolution, accuracy, accuracy condition) since they are not subject to updates or ownership transfer. For every interaction class identified in the Object Interaction Table, the full set of parameters associated with that interaction class must be described in the Attribute/Parameter Table.

The public attributes of objects must be specified in order to support subscription to their values by other interested members of a federation. Thus, the names of attributes and associated object classes are essential information for the RTI. Knowledge of public attributes is commonly required for effective communication between federates in a federation. In addition, while the resolutions, accuracies, and update policies of attributes represent characteristics that are not directly utilized by the RTI (as defined by the HLA Interface Specification), all are important to ensuring compatibility between federates in a federation. A federate operating with very low resolution, accuracy, or update rates for an attribute that it is publishing could create problems for interacting federates that are operating at higher resolutions, accuracies, or update rates. The specification of resolutions, accuracies, and update rates in an HLA FOM is a part of the FOM "contract" between federates to interoperate at the specified levels. It helps ensure a common perception of the simulation space across federates in a federation, aiding the avoidance of inconsistency between federates.

3.3.2 Table Format

The Attribute/Parameter Table of a FOM is designed to provide descriptive information about all public attributes represented in a federation. In addition, it is used to capture information about interaction parameters. The template for the Attribute/Parameter Table is provided by Table 3-5. See Appendix A for a brief description of the syntax used for specifying entries in this table.

Object/ Interaction	Attribute/ Parameter	Data- type	Cardi- nality	Units	Resolution	Accuracy	Accuracy Condition	Update Type	Update Condition	T/A	U/R
										<ta>	<ur>
...	...	...	[<size>]	... <units>	...	...	...	<type>	... <rate> |	... <ta>	... <ur>
			[<size>]	<units>				<type>	<rate> |		
<class> | <interaction>	<attribute> | <parameter>	<datatype>	[<size>]		<resolution>	<accuracy>	<condition>		<condition>	<ta>	<ur>
	... <attribute> | <parameter>	... <datatype>	[<size>] [<size>]	... <units>	... <resolution>	... <accuracy>	... <condition>	<type>	... <condition> <rate> |	... <ta>	... <ur>
			[<size>]	<units>				<type>	<rate> |		
... <class> | <interaction>	... <attribute> | <parameter>	... <datatype>	[<size>]	...	... <resolution>	... <accuracy>	... <condition>	...	... <condition>	... <ta>	... <ur>
	<attribute> | <parameter>	<datatype>	[<size>]	<units>	<resolution>	<accuracy>	<condition>	<type>	<condition> <rate> |		
			[<size>]								
<class> | <interaction>	<attribute> | <parameter>	<datatype>			<resolution>	<accuracy>	<condition>		<condition>		

Table 3-5. Attribute/Parameter Table

The first column, Object/Interaction, lists an object class name for attributes or an interaction class name for interaction parameters. The classes can be chosen from any level of generality in the class structure hierarchy. In general, it will reduce redundancy if attributes are specified for classes at the highest point in the hierarchy to which they generally apply, although this is not required. For example, if all air vehicles have an attribute of minimum turn radius at maximum speed, then it will avoid some redundancy if this attribute is specified just once for the entire class of *Air_Vehicle*. Given that all object subclasses inherit the attribute types of their superclasses, the subclasses of *Air_Vehicle*, such as *Fixed_Wing* and *Rotary_Wing*, also have this attribute with its specified characteristics. When a subclass requires a revision to any inherited attribute characteristic, a new attribute must be defined for the subclass with the required characteristics.

The second column, Attribute/Parameter, lists the public attributes of the specified object class or the parameters of an interaction. The names assigned to attributes of any particular object class must be defined via the ASCII character set, and cannot duplicate (overload) the names of attributes of any higher-level superclasses. There may be many public attributes for a single object class and there may be many parameters for a single interaction class.

The Datatype column is used to reference the datatype of the attribute or parameter. This datatype can be chosen from the list of permissible base attribute/parameter types (as described in Appendix B), or it can be a user-defined datatype. User-defined datatype names must be different than (not overload) the names of the base attribute/parameter types. The specific entry in this datatype column may only contain the name of one of the base attribute/parameter types or an identifier from one of the supplementary tables for enumerated and complex datatypes. When a complex attribute or parameter consists of a homogeneous array or sequence of items which share a common datatype, then this common datatype may be recorded in the datatype column. When the subtypes of a complex datatype are heterogeneous, they require use of the supplemental Complex Datatype Table, as described in Section 3.3.5.

The Cardinality column is used to record the size of an array or sequence. A designation of *1+* in this column allows for unbounded sequences, while fixed integer values designate complex datatypes of fixed length. Cardinalities of multi-dimensional arrays should include the sizes of every dimension listed in their normal order of precedence. For primitive attributes and parameters having only a single element, a one (1) should be entered in this column.

The Units, Resolution, Accuracy, and Accuracy Condition columns are not applicable if the datatype for an attribute or parameter is either enumerated or both complex and heterogeneous. The reason is that these classes of information are either unnecessary (for enumerated datatypes), or are recorded for the individual fields of complex datatypes in the Complex Datatype Table. For these and other datatypes in which units, resolution, and accuracy information do not apply (e.g., strings), the designator N/A for “Not Applicable” should be entered.

The Units column contains the units (e.g., m, km, kg) used for each attribute or parameter whenever such units exist. Any units entered in this column specify the units of the entries in the Resolution and Accuracy columns that follow it.

The Resolution column may have different kinds of entries, depending upon the kind of attribute or parameter. For attributes or parameters of scalar numerical measures, the resolution column may contain a single dimensioned numeric entry for each row of the table. This value may specify the smallest resolvable value separating attribute values that can be discriminated. However, when such attributes or parameters are stored in floating point datatypes, their resolution so defined might vary with the magnitude of the attribute value. Hence, in these cases and others, a better sense of the resolution may be conveyed by the datatype.

The Accuracy column is intended to capture the maximum deviation of the attribute or parameter value from its intended value in the federate or federation. This is ordinarily expressed as a dimensioned value, but may also be *perfect* for many discrete or enumerated attributes. The Accuracy Condition column contains any conditions required for the given accuracy to hold in a given simulation or federation execution. It may consist of reference to a particular type of update algorithm that determines the accuracy, or may be an unconditional *always.*

The Update Type and Update Condition columns record the update policies for an attribute. The update type can be specified as *static, periodic,* or *conditional.* When the update type is *periodic*, then a rate of number of updates per time-unit can be specified in the Update Condition column. Attributes with a *conditional* update type may have the conditions for update specified in the update condition column. For interaction parameters, the indicator *N/A* should be entered into each of these columns.

The Transferable/Acceptable (T/A) column is handled somewhat differently for simulations and federations. In a federation, if an attribute is transferable from a federate, it must be acceptable by some federate in the federation. But a single federate may be able to transfer ownership of an attribute without being able to accept hand-off of attribute ownership from another federate. The basic alternatives for the Transferable/Acceptable column are as follows:

<u>Transferable (T):</u> a federate is currently capable of publishing and updating attributes of the type specified for the object class, and can transfer ownership of the attribute to another simulation using the HLA RTI ownership management services.

Acceptable (A): a federate is currently capable of accepting ownership of this attribute from another federate, including the capability for meaningful continuation of attribute updates.

Not transferable or acceptable (N): a federate is not currently capable of either transferring ownership of this attribute to another federate or accepting ownership of this attribute from another federate.

For an attribute of a SOM, the transferable/acceptable variable <*ta*> may take any of the values from the set {T, A, TA, N}. In a FOM, the only valid entries in this column for federation attributes are *TA* or *N*. For specification of object interaction parameters, this column should contain the indicator *N/A*.

The Updateable/Reflectable (U/R) column of an Attribute/Parameter Table is used to identify the current capabilities of a federate with respect to attribute updating and reflection. Two basic categories are used to indicate capabilities with respect to a given attribute:

Updateable (U) - the federate is currently capable of publishing and updating attributes of the type specified for the object class specified using the *Publish Object Class* and *Update Attribute Values* services of the RTI.

Reflectable (R) - the federate is currently capable of accepting changes to this type of attribute for objects in the specified object class for values provided from calls to the *Reflect Attribute Values* service from the RTI.

For an attribute of a SOM, the updateable/reflectable variable <*ur*> in the Attribute/Parameter Table may take any of three different combinations of capabilities for updating and reflecting, as designated by their abbreviations {U, R, UR}. In a SOM, any listed attribute must be either updateable or reflectable or both. For federations, the appropriate entry should always be *UR* since all attributes in a FOM should be both updateable and reflectable. This column should always contain the *N/A* indicator for interaction parameters.

3.3.3 Inclusion Criteria

All attributes that are designated as public, i.e., whose values are accessible to other federates in a federation, should be documented in the Attribute/Parameter Table of a FOM. All attributes that can be either updated or reflected by an individual federate belong in the Attribute/Parameter Table of its SOM. All parameters to interactions that appear in the Object Interaction Table should appear in the Attribute/Parameter Table. If an interaction parameter is also an attribute of an object class, then it should appear in the Attribute/Parameter Table separately as an attribute and as a parameter.

In some object model descriptions, it may be desirable to document the capability or intent to transfer the privilege of deleting the instantiation of a particular object class from one federate to another. In this case, the attribute “privilegeToDeleteObject”, which is automatically created by the RTI when instantiating an object, should be included in the Attribute/Parameter Table to document the applicable transferability characteristics. If omitted from the table, this privilege is assumed to be neither transferable or acceptable.

3.3.4 Example

Table 3-6 shows illustrative examples of attributes and parameters from the restaurant application as described in Section 3.1.4. In the first entry, the *Employee* object is characterized according to the four attributes shown in the table. The datatypes specified for each of the first three attributes were selected from the list of attribute/parameter basetypes (Appendix B), while the datatype of the fourth attribute is user defined. As with all user-defined datatypes, the indicator *N/A* is placed in the Units, Resolution, Accuracy, and Accuracy Condition columns. Each of these four attributes is updated conditionally except for the *Years_of_Service* attribute, which is updated periodically (yearly) on the employee’s start date anniversary. The Update Condition column for the *Pay_Rate* attribute is annotated with an explanatory “note” as described earlier in Section 3. As with all of the attributes and parameters shown in this example, the attributes of *Employee* are assumed transferable, acceptable, updateable, and reflectable.

The *Waiter* subclass of *Employee* is shown with three attributes. These are in addition to the four inherited attributes from its superclass. Each of the first two attributes, *Efficiency* and *Cheerfulness*, is intended to represent a numeric score (performance measure), that is assigned to the employee at yearly performance reviews. The third attribute is intended to represent the state of the employee (the task he/she is performing) at any given point in time during restaurant operations. The characterization of this attribute is via an enumerated datatype which is described in a separate table.

The next set of entries represents the parameters associated with the interaction class *Food_Arrives_at_Customer*. In this case, two of the three parameters are user-defined datatypes. Since the Units through the Accuracy Condition columns do not apply for user-defined datatypes, and the final four columns do not apply for interaction parameters, only the Datatype and Cardinality columns have entries for these first two attributes. The third parameter uses a boolean datatype (yes or no) to reflect whether the meal was served in a reasonable amount of time.

Object/ Interaction	Attribute/ Parameter	Data-type	Cardi-nality	Units	Reso-lution	Accuracy	Accuracy Condition	Update Type	Update Condition	T/A	U/R
Employee	Pay_Rate	Float	1	Cents/ Hour	1	perfect	always	condi-tional	Merit Increases [1]	TA	UR
	Years_of_ Service	Short	1	Years	1	perfect	always	periodic	1/year, on Anniversary	TA	UR
	Home_ Number	String	1	N/A	N/A	N/A	N/A	condi-tional	Employee Request	TA	UR
	Home_ Address	Address_ Type	1	N/A	N/A	N/A	N/A	condi-tional	Employee Request	TA	UR
Waiter	Efficiency	Short	1	N/A	1	perfect	always	periodic	Performance Review	TA	UR
	Cheerfulness	Short	1	N/A	1	perfect	always	periodic	Performance Review	TA	UR
	State	Waiter_ Tasks	1	N/A	N/A	N/A	N/A	condi-tional	Work Flow	TA	UR
Food_Arrives_ at_Customer	Accuracy_ OK	Boolean	1	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Temperature_ OK	Temp_Type	1	N/A	1	N/A	N/A	N/A	N/A	N/A	N/A
	OK	Accur_Type		N/A		perfect	always	N/A	N/A	N/A	N/A

Attribute/Parameter Table

Table 3-6. Attribute/Parameter Table - SOM Example

3.3.5 Attribute/Parameter Table Subcomponents

3.3.5.1 Purpose/Rationale

While the Attribute/Parameter Table provides columns for datatype specifications, it does not provide definitive guidance for specifying complex datatypes. This section describes additional table formats for complex datatypes as well as for enumerated datatypes to better document their structure and content. These tables are mandatory in situations where a federation or federate implements the attribute or parameter datatypes for which the tables are designed.

3.3.5.2 Enumerated Datatype Table

Table 3-7 describes the format of the Enumerated Datatype Table. The first column defines the identifier (or name) for the enumerated datatype, while the second column provides the specific enumerated values that the identifier can assume. For instance, one potential identifier for an enumerated datatype might be *affiliation*, with the values of *red*, *blue*, and *neutral* representing valid enumerators. The Representation column of the Enumerated Datatypes Table allows the federation to define the agreed-upon numerical value for the specific enumerators. Each identifier name should appear as an entry in the Datatype column of the OMT Attribute/Parameter Table, as was discussed in Section 3.3.2. See Appendix A for a brief description of the general format used for specifying the types of entries in this table.

Enumerated Datatype Table		
Identifier	**Enumerator**	**Representation**
<datatype>	<enumerator>	<integer >
	. . .	. . .
<datatype>	<enumerator>	<integer>
	. . .	. . .
. . .	. . .	. . .

Table 3-7. Enumerated Datatype Table

An example of the use of the Enumerated Datatype Table is provided in Table 3-8. Here, the user-defined Waiter_Tasks datatype specified in the earlier Attribute/Parameter Table example (Section 3.3.4) is characterized according to five different enumerations. Each enumeration represents a state that a waiter can be in at any particular point in time during restaurant operations. The numerical representation of the enumerations does not have to be given in any particular order, but does need to be documented to avoid inconsistent representations among different federates in a federation.

Enumerated Datatype Table		
Identifier	**Enumerator**	**Representation**
Waiter_Tasks	Taking_Order	1
	Serving	2
	Cleaning	3
	Calculating_Bill	4
	Other	5

Table 3-8. Enumerated Datatype Table - SOM Example

3.3.5.3 Complex Datatype Table

Table 3-9 illustrates the format for the Complex Datatype Table. In the first column, Complex Datatype, is the identifier, or name, of the user-defined complex datatype. Complex data type identifiers should match a datatype entry from either the Attribute/Parameter Table or from the Complex Datatype Table itself. The next column, Field Name, provides the means to identify each individual field within the complex datatype. For instance, a complex datatype representing location (with *Location* as its identifier) might have three sub-rows with the field names of *X*, *Y*, and *Z* (for rectangular coordinates). Alternately, two sub-rows with the field names of *Lat* and *Long* might be used. The actual specification of the fields associated with a particular identifier is entirely driven by the requirements of the federate or federation.

The remaining six fields in the Complex Datatype Table are identical to the corresponding columns in the Attribute/Parameter Table (Section 3.3.2). The intent is to capture these classes of information for each field within the complex data structure. This allows certain characteristics common to all fields of a complex attribute (update type/condition, transferable/acceptable, updateable/reflectable) to be specified at the composite level, while characteristics distinctive of the individual fields of an attribute (units, resolution, etc.) are specified at this lower level.

The Complex Datatype Table may also include the names of other complex datatype identifiers within the Datatype column for individual field names. This allows users to build

Complex Datatype	Field Name	Datatype	Cardinality	Units	Resolution	Accuracy	Accuracy Condition
<complex datatype>	<field>	<datatype>	<size>	units	resolution	accuracy	condition
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	<field>	<datatype>	<size>	units	resolution	accuracy	condition
	...	...	...	...	...	...	...
...							
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	<field>		<size>	units			

Table 3-9. Complex Datatype Table

"structures of data structures" according to the needs of their federate or federation. See Appendix A for a brief description of the general format used in specifying the types of entries permitted in this table.

An example of the use of the Complex Datatype Table is provided in Table 3-10. The first complex datatype (*Address_Type*) is shown as consisting of four fields, each identified as an *String* datatype. Each of the other two complex datatypes (*Temp_Type* and *Accur_Type*) consists of three *Boolean* fields. The intent is to specify for each *Main_Course* (composed of one *Entree* and two instances of *Side_Dish*) whether the waiter served exactly what the customer ordered (*Accuracy_OK* parameter) and whether the food was the right temperature (*Temperature_OK* parameter). This information is used by the receiving object in the *Food_Arrives_at_Customer* interaction to determine the value of the customer attribute *Satisfaction*.

Complex Datatype	Field Name	Datatype	Cardinality	Units	Resolution	Accuracy	Accuracy Condition
Address_Type	City	String	1	N/A	N/A	perfect	always
	Street	String	1	N/A	N/A	perfect	always
	Zip	String	1	N/A	N/A	perfect	always
	State	String	1	N/A	N/A	perfect	always
Temp_Type	Entree	Boolean	1	N/A	1	perfect	always
	Vegie_1	Boolean	1	N/A	1	perfect	always
	Vegie_2	Boolean	1	N/A	1	perfect	always
Accur_Type	Entree	Boolean	1	N/A	1	perfect	always
	Vegie_1	Boolean	1	N/A	1	perfect	always
	Vegie_2	Boolean	1	N/A	1	perfect	always

Table 3-10. Complex Datatype Table - SOM Example

4. FOM/SOM LEXICON

4.1 Purpose/Rationale

If interoperability between simulations is to be achieved, it is necessary not only to specify the classes of data required by the templates above but also to achieve a common understanding of the semantics of this data. The FOM/SOM Lexicon provides a means for federations to document the definitions of all terms utilized during construction of FOMs, and for individual federates to document the definitions of all terms provided in their SOMs.

Federations may want to develop additional views on FOM and/or SOM data besides simple term definitions and those explicitly defined by the OMT tables. The absence of additional data views in this document is not meant to constrain federation or simulation developers from defining whatever data views make sense for their specific application. Rather, by providing federation/simulation developers maximum flexibility in this regard, libraries of reusable data views (and automated tools that support them) may be constructed and made available for general use in future applications.

4.2 Table Formats

4.2.1 Object Class Definitions

This section describes the format for defining the object classes that are specified in a given FOM or SOM. A simple template for describing this information is provided in Table 4-1. The first column of this table should contain the names of all object classes described in the FOM or SOM, with the second column describing the semantics for that class. Abstract, higher-level superclasses of instantiable subclasses should be defined as such, along with their purpose in the object class hierarchy. Object classes that can have direct instances (concrete classes) should provide a description of the real-world entity the class is intended to represent, along with any additional information required to clarify the semantics of the class (e.g., fidelity). Users may optionally include the names of the attributes of the object class, and the interactions that the class can initiate or be affected by in the textual description of that object class.

Object Class Definitions	
Term	**Definition**
<term name>	<term definition>
<term name>	<term definition>
. . .	. . .
<term name>	<term definition>

Table 4-1. Object Class Definitions

4.2.2 Object Interaction Definitions

This section describes the format for defining the interactions that can occur between public object classes in the FOM, and interactions that can be published and/or reflected at the individual simulation level in the SOM. The structure for describing this information is provided in Table 4-2. The first column of this table should contain the name of each interaction class. The second column should provide sufficient descriptive information about the interaction class to ensure that the semantics are clearly understood. For abstract interaction classes, this should include the rationale for the use of the class in the interaction class hierarchy, and (optionally) the list of lower-level subclasses it supports. For concrete (instantiable) interaction classes, the definition should include a description of the real-world event the interaction class is intending to represent. The names of the initiating and receiving objects associated with the interaction, and the parametric information that must be included with the interaction, may also be provided.

Object Interaction Definitions	
Term	**Definition**
<term name>	<term definition>
<term name>	<term definition>
. . .	. . .
<term name>	<term definition>

Table 4-2. Object Interaction Definitions

4.2.3 Attribute/Parameter Definitions

This section describes the format for defining the attributes that characterize public object classes and parameters that characterize interactions. The structure for describing this information is provided in Table 4-3. The first column of this table should contain the name of the object class that a given attribute belongs to, or the interaction a given parameter is associated with. This information is useful for associating attributes with object classes, but is also required to distinguish between attributes that share a common name but reside in different classes. The second column of this table should contain the name of the attribute or parameter. The third column of this table should describe the specific characteristic of the object class or interaction that this attribute or parameter is designed to measure. Characteristics of the attribute/parameter that are described in the OMT Attribute/Parameter Table (units, resolution, update rate, etc.) may be repeated in the definition if it clarifies the meaning and purpose for the term.

Attribute/Parameter Definitions		
Class	**Term**	**Definition**
<term name>	<term name>	<term definition>
<term name>	<term name>	<term definition>
. . .	. . .	. . .
<term name>	<term name>	<term definition>

Table 4-3. Attribute/Parameter Definitions

Appendix A: Table Entry Notation

The OMT table specifications for the Object Class Structure Table, Object Interaction Table, and Attribute/Parameter Table use a subset of Backus-Naur Form (BNF) [NAUR60] to specify the types of entries that belong in particular table cells. In BNF, the types of terms to be substituted in the table are enclosed in angle brackets (e.g., <*class*>). Optional entries are enclosed in square brackets (e.g., *[(<ps>)]* for the optional Publishable/Subscribable capability entries of the Object Class Structure Table). Any parentheses are terminal characters which should appear as shown. Thus, the basic entry in a cell of the Object Class Structure Table, designated by *<class> (<ps>)*, indicates a class name followed by a Publishable/Subscribable code in parentheses. An asterisk (*) is used to indicate a repetition of zero or more instances, such as in the last column of the Object Class Structure Table where it indicates a variable number of entries for the most specific types of classes, as follows:

```
[<class> (<ps>)]  [,<class> (<ps>)]*  |  [<ref>]
```

A vertical bar (|) is used to indicate alternative possible entries. Thus, the specification for the last column of the Object Class Structure Table (above) indicates optional entries of either a variable length list of classes with Publishable/Subscribable codes or a reference to another table.

Appendix B: Attribute/Parameter Basetypes

The following list defines the complete set of basetypes that may be used to characterize object attributes or interaction parameters.

- float - IEEE single-precision floating point number
- double - IEEE double-precision floating point number
- short - integer value in the range $0…2^{16} - 1$
- unsigned short - integer value in the range $-2^{15}…2^{15} - 1$
- long - integer value in the range $-2^{31}…2^{31} - 1$
- unsigned long - integer value in the range $0…2^{32} - 1$
- char - 8-bit quantity with a numerical value between 0 and 255 (decimal)
- boolean - quantity which can only take one of the values TRUE and FALSE
- octet- 8-bit quantity guaranteed not to undergo any conversion
- any - permits the specification of values which can express any basetype
- string - one-dimensional array of "chars" which is terminated with a NULL (0 value) char
- sequence - one-dimensional array of any basetype with two characteristics: a maximum size (which is fixed at specification time) and a length (which is determined at run time)

Acronyms

ASCII	American Standard Code for Information Interchange
BNF	Backus-Naur Form
DoD	Department of Defense
DMSO	Defense Modeling and Simulation Office
FOM	Federation Object Model
HLA	High Level Architecture
N/A	Not Applicable
OMT	Object Model Template
OO	Object-Oriented
RTI	Runtime Infrastructure
SOM	Simulation Object Model

References

[DOD95] Department of Defense, Under Secretary of Defense (Acquisition and Technology) (USD (A&T)), *DoD Modeling and Simulation (M&S) Master Plan*, Washington, DC, October 1995.

[NAUR60] Naur, P. et al., "Report on the Algorithmic Language ALGOL 60," *Communications of the ACM*, Vol. 6, No. 1, January 1963, pp. 1-17.

Comments

Comments on this document should be sent by electronic mail to the Defense Modeling and Simulation Office HLA Specifications mailing address (hla_specs@msis.dmso.mil). The subject line of the message should include the OMT section number referenced in the comment. The body of each submittal should include (1) the name and electronic mailing address of the person making the comment (separate from the mail header), (2) reference to the portion of this document that the comment addresses (by page, section number, and paragraph number), (3) a one-sentence summary of the comment and/or issue, (4) a brief description of the comment and/or issue, and (5) any suggested resolution or action to be taken.